TM-5898/000/00
30 June 1977

ARC SYSTEM SOFTWARE SPECIFICATION
(CDRL A005)

SPONSORED BY: Defense Advanced Research Projects Agency (DOD)
ARPA Order No. 3037, Amendment 12

MONITORED BY: NAVELEXSYSENGCEN, POW-1, Vallejo
Under Contract No. N00228-77-C-2070
Contract Dates: 3 December 1966 to 30 June 1977
Reporting Period: 30 June 1977

500 Macara Avenue • Sunnyvale, California 94086 • (408) 737-8000

TM-5898/000/00
30 June 1977

ARC SYSTEM SOFTWARE SPECIFICATION
(CDRL A005)

PRINCIPAL AUTHORS

THOMAS H. HINKE (213) 825-7511, X2556
CARL M. SWITZKY (213) 825-7511, X2969

APPROVED BY

[illegible]

SDC/ARC PROGRAM MANAGER, DR. E. LEVIN

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

Task Objectives

The Acoustic Research Center (ARC) is a specialized network of computers that allows for the acquisition of streams of acoustic data from multiple sites and for sophisticated signal processing of the data. Because of the complexity of the ARC multi-computer environment, and the lack of easy to use software support, there are difficulties in describing, testing, combining and executing ARC experiments. These difficulties were outlined in a study done for the ARC by Mitre/Metrek and documented in the ARC user's Interface and Metaexec Design Considerations report [1].

The study proposed that general purpose software be developed to make the ARC a more useful tool for doing acoustic research. The software would aid experiment development and permit the experimenter to easily interface with the ARC: This software included a distributed operating system called the Metaexec, and a user language for describing experiments. The language is currently under development in a parallel effort.

This report represents the first part of a task whose objective is to design and implement a Metaexec which will facilitate the use of the ARC by experimenters. In order to properly address the unique problems at the ARC, the Metaexec must satisfy certain objectives that have been identified in the course of this effort.

The first objective of the Metaexec is that it must control the running of experiments and the allocation of ARC resources for the multicomputer ARC processing environment, in much the same way that a single computer operating systems controls the running of programs and the allocation of its local resources.

The second objective is that the Metaexec must support the experiment description user language, called LINGO, which is currently under development in a parallel effort [3]. This language describes ARC experiments in terms of nodes which perform processing functions (such as FFTS or correlations) on data and links which transport data from one node to the next.

A third objective of the Metaexec is that it must support the real time data input rates provided by the ARC data sources.

A fourth objective of the Metaexec is that it should be able to support a degraded mode of operation if individual processor failures occur. This may include the reassignment of functions to other processors and/or the preservation of certain highly critical experiment functions in the face of major processor failures.

In order to decouple the major portion of the ARC processing system from the requirements of real time data processing and to provide a means of repeating experiments on selected data, a fifth objective is that the system should capture all data flowing into the central site.

A sixth objective of the Metaexec is that it must be capable of handling system expansion without major system impact.

A final Metaexec objective is that it must be capable of supporting the performance of several independent experiments simultaneously through the sharing of data and resources. It must allow separate starting and stopping of experiments and support a varying experiment mix.

Technical Problems

The primary technical problem encountered thus far in the design is the necessity of providing a modular, expandable, reliable system capable of handling arbitrary experiment network configurations while still maintaining the ability to process real time data at fairly high potential data rates. An important factor in the solution of this problem has been the decoupling of experiments from real time data variability through the use of the initial data capture upon entry into the central site.

General Methodology

The methodology employed in the ARC system software design effort involved gaining an understanding of ARC processing requirements, the synthesis of the concrete objectives to be met by the ARC system software to satisfy these requirements, and a top down, structured design approach to carry the system objectives into software functional modules. These various facets of the methodology were accomplished through the following activities:

1) Review of current ARC system and experiment software.
2) Discussions with ARC experimenters and ARC system software experts.
3) A review of studies concerned with advanced ARC development.
4) A review of requirements document, prepared for the ARC.
5) A review of pertinent literature on multicomputer, distributed processing systems.
6) An identification of system objectives.
7) A synthesis of ARC system software functions from the objectives identified.

Technical Results

The accompanying report presents a Top Level Metaexec design which is capable of supporting ARC experiments described in a high-level node/link language. The node/link language views an experiment as a network of processing stations (called nodes) connected by data conduits (called links). The Metaexec is the multicomputer operating system which maps the concept of a node and link into the different operating systems provided by the ARC processors. Conceptually, the Metaexec sits on top of the various local ARC operating systems and provides a multicomputer, ARC experiment oriented Meta operating system which permits an experimenter to utilize all of the capabilities of the ARC processing resources by describing his task in terms of user-defined and systems-provided processing nodes and associated links.

In order to ensure the efficient and reliable operation of the Metaexec during an experiment, its run time experiment control functions have been distributed among the main ARC processors. Each main ARC processor has a complete set of run time Metaexec functions. These distributed functions, which comprise what is termed the Local Metaexec, support that portion of an experiment's nodes and links which run on its processor.

Each Local Metaexec is designed around a centralized, standard internode linking mechanism which serves as an internal data bus for those functional modules which may be changed from experiment to experiment to support either different processing requirements or increased capability. Such increased capabilities may be due to the addition of added ARC processors or access to off-site processing resources such as the ILLIAC IV or the ARPA net.

Since the Local Metaexec is designed around the anticipation of future added capability, these processors or interfaces can be added with little perturbation to the Metaexec software then existing.

In addition to the Local Metaexec, the design also recognizes the need for centralized monitoring and management of the ARC processing resources in order to provide for the necessary tuning of ARC resource allocation to efficiently support the maximum experiment capability. This centralized monitoring capability, which is provided by the central Metaexec (CME), is also used to alert the ARC director of failures within the ARC processing environment and to provide failure analysis functions with the necessary data needed to provide recommendations on desirable alternative processing configurations or feasible modes of degraded operation. The design of the CME has anticipated the possible failure of its processors, and provides means by which a backup CME can be activated at another processor. In addition, the central Metaexec, through support provided by the Local Metaexecs provides a debugging capability which assists the experimenter in testing this experiment's node/link network prior to utilizing it for live experimental data.

Important Findings and Conclusions

The basic architecture of the Metaexec represents a very important result of the ARC system software development effort. The architecture is well suited to the efficient scheduling and coordination of both single ARC experiments and multiple ARC experiments, while maintaining an architecture which is flexible enough to permit changes in the experiments network topology as well as changes in the ARC resource topology. The architecture makes it very convenient to "plug in" interface modules for anything from a new parallel processor to a new Host on the ARPA net. These two dimensions of both experiment and ARC resource modularity and reconfigurability provide the flexibility necessary to support the ARC's rapidly changing and somewhat unknowable current and future processing requirements.

Implications for Further Study

The development effort is on-going and hence further research areas will be more readily identifiable at a later date in the project.

Significant Hardware and Software Developments

The important technical results have been reported above. No operational software or hardware has yet been developed.

1.0 INTRODUCTION

1.1 SCOPE

This report will present a top level design of the system software for the Acoustic Research Center (ARC) of the Defense Advanced Research Projects Agency. The report will describe the Metaexec in terms of the functional decomposition, its utilization on multiple processors, and its relatio. ıp to the user language.

The ARC is a specialized network of computers that allows for the acqu n of streams of acoustic data from multiple sites and for sophisticated signal processing of the data. Real time acoustic samples are collected and pre-processed at each of the remote sites by a set of general and special purpose computers, and transferred to a central site.

The central site provides data recording facilities, comprehensive signal processing capabilities through additional general and special purpose computers,and mechanisms for transmitting data to the ILLIAC IV parallel processing array and to other ARPANET host computers. A detailed description of the current ARC system can be found in the ARC User's Manual [4].

Because of the complexity of the ARC multi-computer environment, and the lack of convenient software support, there are difficulties in describing, testing, combining, and executing ARC experiments. These difficulties were outlined in a study done for the ARC by Mitre/Metrek and documented in the ARC User's Interface and Metaexec Design Considerations report [1]. The study proposed that some general purpose software be developed to make the ARC a more useful tool for doing acoustic research. The software would aid experiment development and permit the experimenter to easily interface with the ARC. This software included a distributed operating system called the Metaexec, and a user language for describing experiments.

The experiment description language is a special purpose language with which ARC users would be able to conveniently express their experiments, and then use these descriptions as input to the Metaexec to run experiments. The language is described in detail in the Preliminary Report on the Network Language LINGO [3]. The ARC Metaexec will take an experiment description written in LINGO, set up the appropriate mechanisms in the various processors, and control execution of the experiment. This design of the top level Metaexec software will be described in this document.

This report is organized into four chapters. The remainder of this chapter will discuss the general nature of the Metaexec in terms of general ARC requirements, and Metaexec objectives. Section 2 will present the Metaexec system design and functional decomposition, while Section 3 will provide a requirements allocation summary. Section 4 will be reserved for the final report, and will discuss testing and integration of Metaexec software.

1.2 REQUIREMENTS

The ARC may be viewed as a research vehicle for acoustic data processing. This vehicle is utilized by qualified users who propose, develop, and perform acoustic experiments. These experiments require and utilize resources at the ARC such as data collection facilities, data processing facilities, and data storage facilities.

As acoustic research progresses, experiments become more sophisticated and require more resources. This has led to an increasingly complex ARC configuration which in turn has led to difficulties in describing and performing the experiments. It is the overall goal of the ARC Advanced Design to alleviate these difficulties.

In order to drive an advanced design such as at the ARC it is important to develop a set of requirements. With the experimental nature of the ARC, this task can be partially addressed by analyzing the requirements of existing and proposed experiments, and using these to generate a set of general requirement characteristics. It should be noted here that the union of all requirements for proposed experiments may be an unrealistic as a specific goal, but is very useful for deriving general requirements for an advanced design.

The analysis of existing and proposed experiments was performed and is given in the ARC Requirements Analysis Final Report [2]. These requirements have led to specific objectives for the design of the Metaexec which will be presented in the next section.

1.3 METAEXEC OBJECTIVES

The term "Metaexec" is the designation given to a large portion of ARC system software. This software can be viewed as a multi-computer operating system that controls the running of experiments and the allocation of ARC resources much the same way a single computer operating system controls the running of programs and the allocation of its local resources. The Metaexec objectives presented in this section are based on the overall ARC requirements, and have been used as the basic design criteria for the functional decomposition given in Section 2.

1.3.1 LINGO Support

ARC experiments are described by the experimenter to the system in terms of nodes and links. Nodes perform the data and signal processing operations that are needed for the experiment. Links are the mechanism by which nodes are connected to each other. The Metaexec must utilize the resources provided by each of the ARC processors to provide an environment that will support the nodes and links called for in a LINGO experiment description.

Included in this support is the proper execution of the semantics that are part of LINGO such as rules for node invocation, and link behavior.

One way of viewing this objective is to perceive each of the processors containing certain resources such as programs, storage, and special processing hardware. These resources can be combined together to produce the abstract resources of nodes and links. The LINGO language gives precise definition of the different attributes of nodes and links, how they can be connected, and what the behavior of a node-link network should be. Thus LINGO defines an abstract machine which will execute node-link networks. It is the objective of the Metaexec to provide an implementation of the abstract LINGO machine. This means that the Metaexec has effectively two tasks. First, it must utilize the processor resources to form the nodes and links required by an experiment, and secondly, it must manipulate the resources so that the experiment behavior follows the LINGO semantics. In these ways, the Metaexec will support LINGO.

1.3.2 Real Time Processing

Many experiments within the ARC require that the processing be carried out so that results are available within a short time of receiving the input, in order that subsequent processing may be effected. This requirement is reflected at various points of processing throughout the ARC. At remote sites, for instance, the processing time for a sample of data must not be greater than the collection time for that sample. If the processing time is greater, then the real time requirements will not be met because either a backlog data queue will build up, or data will be lost. At the central site, it will usually be desirable to provide a spool on which all incoming data may be stored for a period of time. Because the remote sites operate with a real time constraint and may only have limited storage capacity, the central site must accept and record the incoming data within the same real time constraint. Subsequent handling of data by the experiment may also require real time processing capability because of the need for current results or control feedback loops. These real time experiments require that all the necessary data be processed within a time frame that is adequate to support the experiment. The Metaexec must be able to support this real time processing requirement. It must be noted, however, that any computer configuration has a maximum processing capability that cannot be exceeded. If an experiment or group of experiments attempt to exceed the system's capability, it will be impossible for the Metaexec to maintain the real time processing rates. Thus the Metaexec must monitor the processing loads of the system and inform ARC users when real time constraints cannot be met. The Metaexec must also support the ability for an experiment to run faster than real time from spooled data when system processing capability is available.

1.3.3 Multiple Computer Support

The ARC consists of a heterogeneous network of computers that are interconnected through a wide variety of interfaces. These interfaces include direct hardware connections, shared I/O devices, satellite connections, and the ARPANET. Because the Metaexec must be able to support experiments in a distributed manner on the ARC, it must be able to determine which processors are necessary or suitable for each node of an experiment and how each required link will be established. During execution of the experiment the Metaexec must cause appropriate actions to occur at each processor so that the multiple computer operation and coordination take place correctly.

1.3.4 Degraded Operation During Partial System Failure

The Metaexec should be able to support a degraded mode of operation if individual processor failures occur, and no single component failure should disable the entire system.

When a hardware failure occurs it should be possible to reassign the functions that were being performed by that processor to other equipment, so that processing may continue. This objective requires that the ARC equipment be configured so that there are no critical paths within the system that are susceptible to failure. It also implies that any Metaexec software that is critical to the entire operation be transportable and operational on other processors.

In the event that a major system failure does occur, the ARC will not be able to support the same level of performance that is possible when the entire system is operational. If this condition occurs, then the Metaexec must provide the capability of controlling which functions will continue, and at what levels of performance.

1.3.5 Central Site Data Capture

Because of the unpredictable nature of data requirements for ARC experiments, it is essential that the ARC provide the capability for capturing all data that flows into the central site. Capturing of data includes both long term archiving and medium term spooling. The long term archive may be used for rerunning experiments or testing new experiments. Experiments might be rerun to verify prior results, to apply new processing procedures, or to utilize hardware resources that were previously unavailable because of either prior allocation or component failure.

The spool is used to decouple subsequent processing from real time, and to allow experimenters to back up on the data stream, and reprocess data. The Metaexec must be able to record all central site input data on both the archive and the spool simultaneously, and at the maximum input data rate for the specific ARC configuration.

1.3.6 System Expandability

The ARC system is seen as a gradually changing computer network, that will start with some preliminary configuration, and increase in processing capability as additional hardware is incorporated into the system. Additions would increase all aspects of the ARC signal processing capabilities including quantity and arrival rate of incoming data, storage and spool capacity for data, and amount of processing power available for handling data. The Metaexec must be capable of handling the system expansion without major system interruption. Its design should not be dependent on any specific hardware configuration and should handle all system expansion in a uniform manner.

1.3.7 Multiple Experiment Capability

The ARC will be used to perform several independent experiments simultaneously through sharing of data and resources. The Metaexec must allow starting and stopping of experiments, and support a varying experiment mix. The Metaexec should provide status information as to the state of the system and the allocation of resources so that the impact of a new experiment on the current experiment mix may be analyzed. The starting and stopping of experiments should be performed by a convenient set of procedures and should cause a minimal amount of disruption to other experiments in progress.

2.0 FUNCTIONAL SPECIFICATIONS

This section provides an initial top level design of the ARC software. Its approach is to describe a first level functional decomposition of the ARC software and then carry two of the first level functions, those which comprise the Metaexec, down to a second level of decomposition. The objective of this section is to construct a complete, logical Metaexec structure which can serve to tie together the more detailed design and implementation which is to follow.

2.1 FIRST LEVEL DECOMPOSITION OF FUNCTIONS

The starting point for a first level decomposition of ARC software is found in the general purpose of the ARC which is to support the running of real time signal processing oriented experiments. Such experiments will be described in the node/link experiment description language LINGO and run on the multicomputer processing environment of the ARC. Within the group of processors required to support ARC experiments can be found two distinct types of processors.* The first class includes those processors which can support the node and link concept of an experiment. They support the standard ARC node/link protocols which are used to realize the experiment network topology specified by the experimenter in LINGO. They will be termed node link processors or, more simply, NL-processors. The second class of ARC processors include those which do not support the node/link concepts or protocols. This may be due to the fact that the processors are very specialized, limited in memory, or not easily programmed such that they can not be readily adapted to node/link support. Alternatively, they may not be under the jurisdiction of the ARC as would be the case with an ARPA net host used in the course of an experiment. Such a processor may not support the concepts of nodes and links with their associated

*A processor consists of a computer and its associated executive.

standard ARC protocols. While a non-NL-processor does not support the node/link concept, it does support the experiment by performing arithmetic computations or other services, on behalf of one or more of the nodes in the experiment. This class of processor will be identified as a non-node/link processor, or simply a non-NL processor.

At this top level of functional decomposition, two functions can be identified. These are the NL-processing functions and the non-NL-processing functions. The NL-processing function will be called the local Metaexec function, since it will be associated with an individual NL-processor. The non-NL-processing functions are experiment specific and will only be described in terms of how they can be linked to an ARC experiment through the local Metaexec. Therefore, all subsequent discussion of non-NL-processors will be under the local Metaexec functions discussion.

At this stage in the presentation, it should be noted that both NL and non-NL processing functions come in two types. The first type includes those processors which are capable of supporting ARC real time processing. The second type are processors which need not support real time processing in the ARC, since they may, perhaps through a large input spool, have been decoupled from real time. While this fact is important for the next level of design, the node design, it is not really pertinent to the current and following discussion of the ARC software.

A third function that can be identified at this level is the experiment development function. Under the umbrella of this function will fall all those ARC systems associated with preparing an experiment for submission to the ARC Metaexec. This function is broken out here to show how it relates to the Metaexecutive, but its main description will be found in those documents dealing with the language.

The final function to be identified at this level results from the realization that some central function will be required to support the various distributed functions previously identified. This central function will be termed the Central Meta Executive.

At this top level of design, four functional areas have been identified. These include the Central Metaexec, the Local Metaexec, Experiment Development, and non-NL-processing. The first three functions are under ARC jurisdiction and will be described in more detail in the following sections.

2.1.1 Experiment Development

An experiment is initially described in the node/link language LINGO. This LINGO description is input to a language processor, called a linker, which validates the correctness of the node connections specified, provides guidance to the user if inappropriate connections are specified, and when correct, outputs a global operations order which describes the entire experiment in terms that can be used by an experiment loader to load and run the experiment. The details of the linker will be described in more detail in the documentation accompanying the language development effort. The important observation to be made at this point is that the linker outputs a global operations order which will be used by the Meta Executive to load and run the experiment. It is this load and run time aspect of the ARC system design which will be more fully addressed in this report. These functions fall under the jurisdiction of the Local Metaexec and Central Metaexec.

2.1.2 Central Metaexec

Two different design concepts were considered when determining the relationship between the Central Metaexec (CME) and the Local Metaexec (LME). The nature of this relationship had a major impact on the character of the CME. The first concept was to consider the CME as a central control which orchestrated the run time control of an experiment in real time. The opposing view, and that which was chosen, was to view the ARC as separate processors which were initially orchestrated at load time, and then self coordinated during run time.

This concept of distributed control was chosen for efficiency and reliability reasons. Since experiment topology can be specified at load time and is fixed until changed by the experimenter through a change in parameters or the addition of new nodes or links, no real need exists for central control of the run time execution of experiment nodes and links. The global view of an experiment, represented by the global operations orders (G-OPS), fully defines and takes into account the global issues of experiment data flow and control. The G-OPS can then be decomposed into the responsibilities that each processor must undertake in support of the experiment as a whole. Once these responsibilities in the form of local operations orders (L-OPS) have been transmitted to each processor, central control no longer needs to be involved in the run time support of node activation, data flow or inter-node control information flow.

This approach has the advantage that a central control computer is not given the opportunity to become a bottleneck for system control. By distributing the central control function, N processors now undertake this responsibility with all the speed advantage provided by parallel processing. In addition, most of the control lines are kept within a single processor rather than having to involve <u>both</u> the local processor and the central processor.

A final justification for distributed control is that single points of failure can be more easily avoided. Even though, as discussed later, the distributed control approach still requires some central experiment support (provided by the CME), the nature of this support is such that it can be more easily relocated than can a totally central control oriented function.

This concept of distributed control does not, as mentioned above, obviate the need for some form of central experiment support. Such support takes two forms. The first is that support required to generate the local operations orders from the global operations orders and then distribute them to the appropriate processors. The second form of central support is to monitor the status of experiments and ARC processors and take appropriate action when required. These central support functions grouped under the Central Metaexec function will each be discussed in turn.

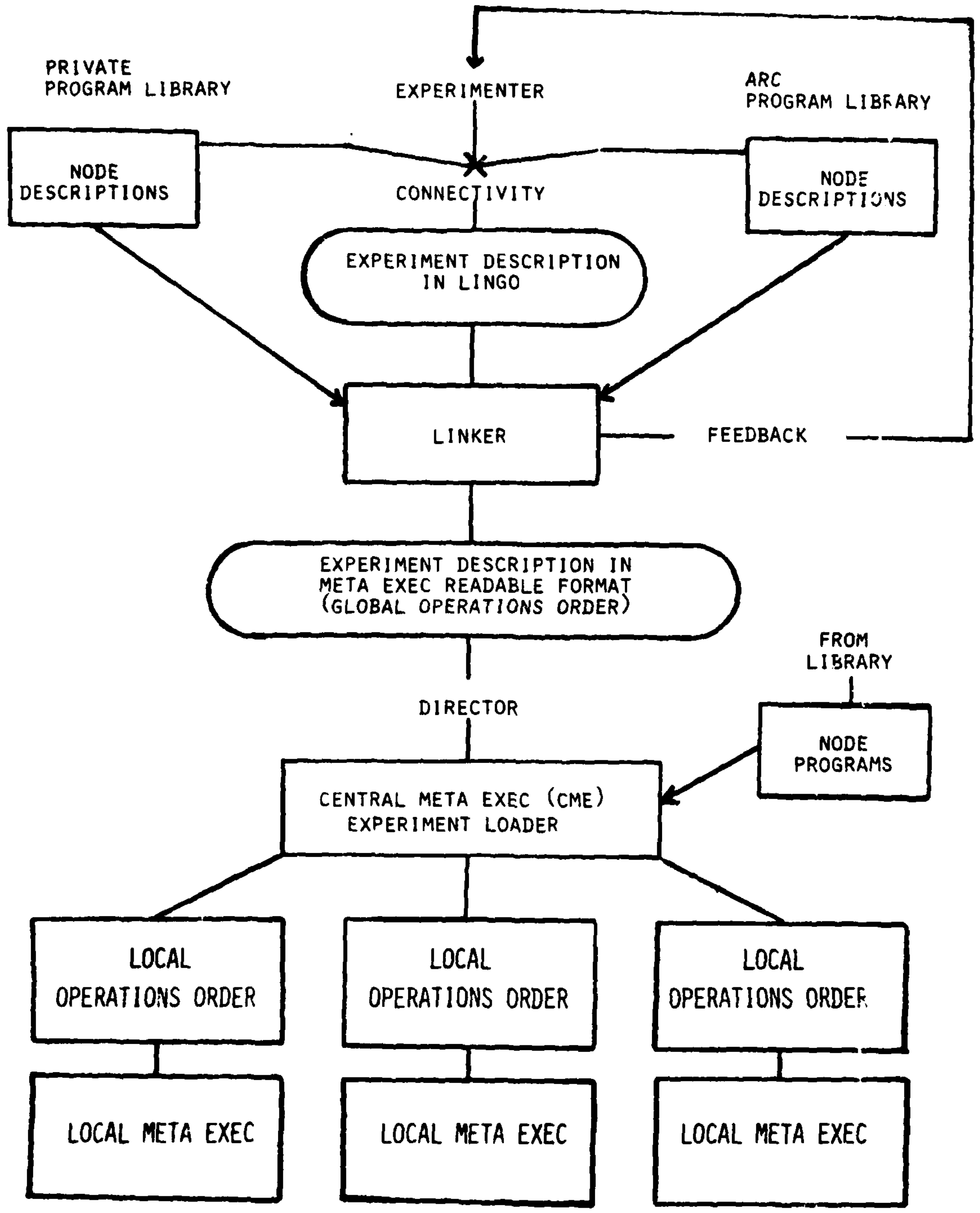
Figure 2-1 illustrates the role that the CME plays at load time. The CME experiment loader inputs the experiment described by the global operations order and decomposes this into each individual processor's responsibility for the experiment. Each ARC NL-processor is then sent a L-OP which describes its role in the experiment. In addition to the L-OPS, the CME loader also insures that the programs associated with each node of the experiment are either resident at the respective processor or are sent to the processor from the program library.

The CME load function is required for not only the initial load of an experiment, but also for subsequent changes in the experiment topology and for experiment reconfiguration in case of a partial system failure. These issues will be discussed at greater lengths during the discussion of the second level functions.

The system monitoring aspect of CME support is required in order to assess the performance of all aspects of an experiment to insure that necessary real time processing rates are maintained. In addition, monitoring is required to continually assess the status of ARC resources and take necessary action to reconfigure an experiment (e.g., shift nodes from one processor to an alternative processor) or preserve important experiment functions (e.g., initial data spooling) during a partial system failure. Hence, the status of ARC physical resources must be monitored and necessary failure analysis and reconfiguration recommendations must be generated to facilitate a proper decision by the ARC director. A final aspect of monitoring is the necessity to provide some system support for experiment debugging. Trace routines, link dump routines, routines to permit the testing of part of our experiment on real or archival data, and other routines must be developed to facilitate the checkout of an experiment.

In addition to the monitoring and loading functions previously discussed, the CME must also provide an interface to the outside world. One interface provided

FIGURE 2-1 THE CME AT LOAD TIME



by the CME is for the ARC director. The ARC director is that individual charged with the responsibility of overseeing the facility during an experiment and mediating requests for shared resources or changes in the operating characteristics of shared resources by the various experimenters concurrently using the facilities of the ARC. The director thus needs some interface through which he can exert control over the processing environment of the ARC experiments. Such control capabilities will be provided by an experiment control function.

A second CME interface is required for the experimenter or user of the system. The interface will be used primarily during system debugging. During the normal running of an experiment, experimenter control will be exerted through terminal and control nodes defined in the topology of the nodes and links of the experiment.

In Summary, the functions identified for the CME in the previous section include the following:

1. Director Interface
2. User Interface
3. Loading
4. Experiment Control
5. Experiment Monitoring and Tuning
6. Failure Analysis and Recommendations
7. Machine Monitoring
8. Debugging

These will be more fully explored in the discussion on the second level functional decomposition.

2.1.3 Local Metaexec

A Local Metaexec (LME) exists on each NL-processor in the ARC. The primary purpose of the LME is to support that portion of an experiment's nodes and links which execute on its respective processor.

Prior to embarking upon a discussion of the nature of the functions provided by the LME, it is important to place the LME into proper perspective by delving briefly into what may be construed as an implementation issue. A NL-proccessor's system software consists of an operating system and the Local Metaexec. The operating system, either supplied by the manufacturer of the computer or provided by an independent vendor, is assumed to provide those services traditionally associated with a modern operating system. These services include support for processes, input/output, a file system, device and programmed interrupts process swapping, prioritized scheduling*, and necessary device interfaces. The LME will be built on top of the abstract machine provided by the processor's operating system. At this level of design, these services of the abstract machine will be assumed, and hence, only those functions unique to the Metaexec will be described.

The experiment support that must be provided by the LME is primarily concerned with node (process) activation, to properly support the progress of the experiment, and link implementation to carry data and control information from one node to the next.

Node activation requires a LME scheduler which may either direct the operating system scheduler or substantially replace it. The LME scheduler schedules nodes

*Prioritized scheduling is required for those LMEs that must interface with real time data collection and may also be needed by the LME scheduler to tune the node activation sequence for more efficient processing.

based on experiment events rather than on priority or hardware resource availability or completion events as does the operating system schedule. Hence, the LME scheduler must act in a master relationship to the operating system scheduler.

The realization of links between nodes, which may be on the same or different processors, requires both a run time intra-processor and inter-processor linker. Both forms of the linker must provide for the efficient transfer of data between nodes in a manner which will adequately support the real time requirements of the system. In addition, inter-processor links must be provided between NL-processors and between NL-processors and non-NL-processors. The former will be provided by the interprocessor linker function and the latter by a gateway function. This gateway will be designed to handle the mapping between standard NL-protocols and the non-NL-protocols.

Support for both link activation and node activation will be provided by tables which at this level of the design will be assumed to be managed in a node/link state function. Such tables will keep track of the state of links (e.g., full or empty) and the state of nodes (e.g., ready to run).

In order to load the experiment descriptions (node/link topology) and node programs into the local processor, the LME must provide a loader which can receive and act on the L-OPS from the CME.

An additional function that must be supported by the LME is that of local control. Commands initiated by the ARC director at the CME are received and acted upon by the LME local control function. These commands include such things as start and stop experiments as well as queries on the status of local resource utilization and requests for performance measurements or debugging assistance. All commands applicable to LME system control are handled by the local control function, and then possibly passed on to functions designed to handle the specific area of concern.

The LME also requires a function to assess the status of experiments and perform performance analyses when so directed by the CME. In addition, the LME must periodically monitor the status of the hardware and relay its findings to the CME for evaluation of total system availability.

In order to facilitate the debugging of experiments, a debugging function is required at the LME. This function will perform those debugging functions previously identified for the CME.

Finally, the LME must support both standard system nodes and user nodes. Standard system nodes are those nodes which are explicitly specified in the experiment by the experimenter, but which require access to some standard and possibly shared device such as a disk, display, printer, or terminal. A spool node, which stores and retrieves real time from the disk would be one example of a shared system node. It would be responsible for queueing disk requests, optimizing disk accesses, locating the desired data based on keys provided by the experimenter, and finally retrieving the data. User nodes are those signal processing nodes which the user either writes himself or has written by the ARC contractor.

The previous discussion has provided a sketch of the nature of and motivation for the LME function. It has resulted in the identification of the following secondary functions:

Scheduler

Run Time Linker (intra-processor)

Inter-processor Linker (standard NL protocols)

Gateway

Node/Link State Function

Local Loader

Local Control

System Status and Performance

Hardware Status

Debugging

System Nodes

User Nodes

Each of these will be explored in more detail in the second level functional decomposition discussion.

2.2 SECOND LEVEL FUNCTIONAL DECOMPOSITION

This section of the report will present more details of the design of each of the second level functions introduced in the course of describing the first level functions. The discussion will first describe the local Metaexec (LME) functions and then the Central Metaexec (CME) functions.

2.2.1 Local Metaexec

Figure 2-2 is a functional diagram of the LME. The first concept that the figure conveys is that the processors indigenous operating system provides the abstract machine basis for all of the functional blocks listed. Those blocks directly attached to the operating system are assumed to require a close relationship with the operating system, including possible modifications to it, in order to properly carry out the task of the LME. These closely coupled functions include the Scheduler, Interprocessor Linker, Run Time Linker, Gateway, Local Loader, and Hardware Status. It may also be necessary to permit such a close association for some of the system nodes as well, but this decision must await further design. The dashed line between the Scheduler and the Operating System indicates the anticipated very close relationship that must exist, and opens the possibility that the Metaexec Scheduler might have to substantially replace the operating system scheduler.

The Figure 2-2 LME software structure also shows a top level view of the module to module interaction. It indicates that the local control function, while performing control functions within itself (e.g., start and stop experiment) also calls other functions to satisfy some of its commands. As a result of appropriate commands, it invokes the system status, debugging, local loader, and hardware status functions.

The design also indicates that all communications between the LME control functions and the outside world arrive through the Run Time Linker via either the Interprocessor Linker or Gateway functions. As indicated on the figure, this control

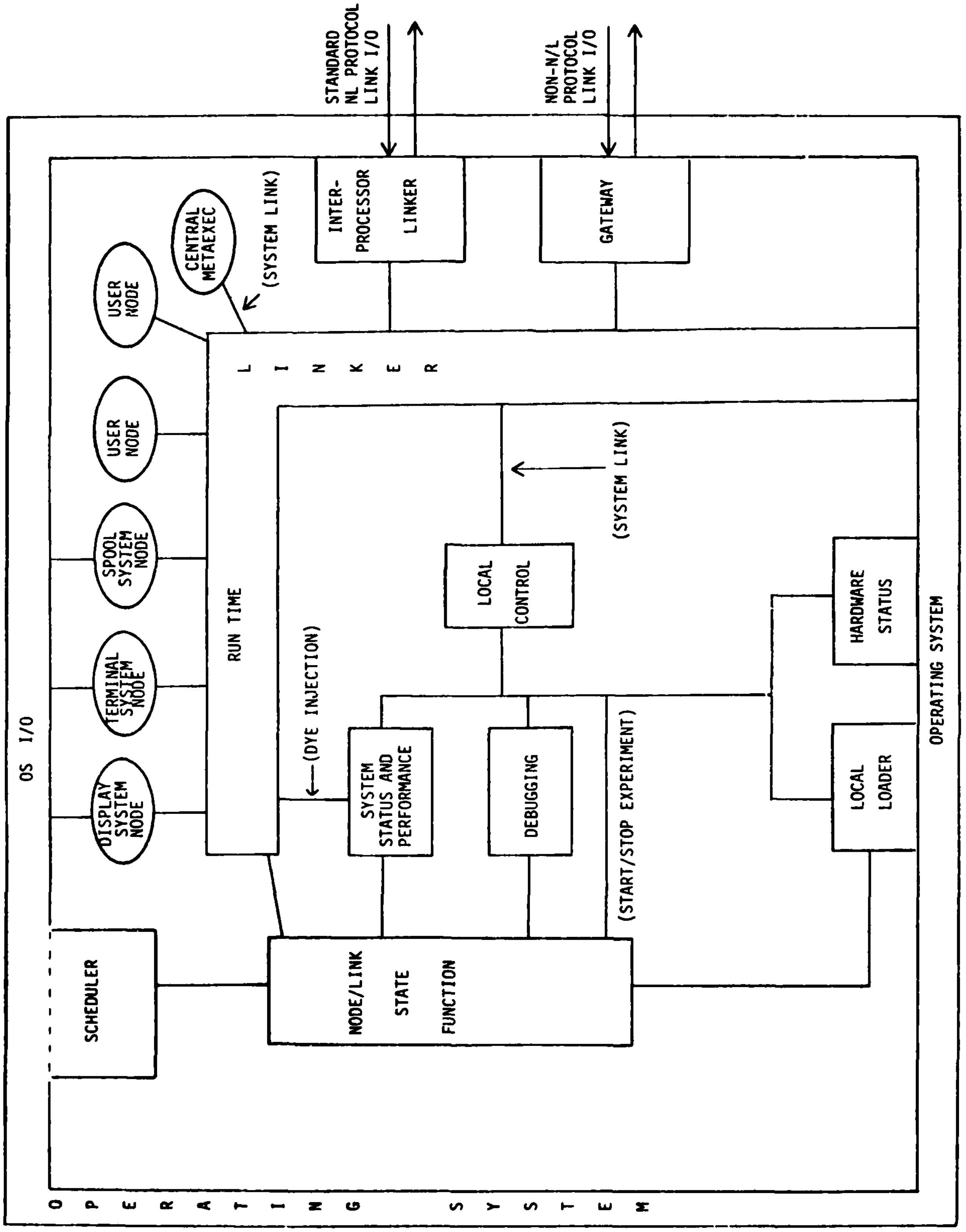


FIGURE 2-2 LOCAL METAEXEC

related data arrives over a system link which is always active while the processor is operating.

One important concept illustrated by Figure 2-1 is the similarity between the two interprocessor functions (Interprocessor Linker and Gateway) and the system nodes (Display, Terminal, and Spool). All are involved with I/O and hence must use the operating system, and all utilize the Run Time Linker as an internal bus to transport data between each other and user nodes requiring the data. At this point, the primary distinction between system functions and system nodes is that the system nodes are visible to and specified by the experimenter designing the network, whereas, the interprocessor system functions are invisible to the experimenter. A further difference may be apparent in implementation, since the interprocessor functions may be implemented as operating system functions while the system nodes may be implemented as processes. For these reasons, the distinction will be maintained, but the similarity should be kept in mind as the design progresses toward implementation.

A final function illustrated in Figure 2-2 is the Central Metaexec. It resides on the LME and uses the Run Time Linker to communicate its commands to the local control on its own LME as well as the local control of all other ARC NL-processors.

With the previous overall LME system concept in mind, the discussion will now explore design details of each of the functional areas in Figure 2-2. The CME function discussion, however, will be delayed and handled in a separate section following the LME discussion.

2.2.1.1 Node Introduction

In order to fully understand the approach taken in some of the other LME functions, the nature of the node (either system or user) should be understood. As mentioned previously, the node is viewed as a process. A process is the activation of a program, and it is anticipated that a single program may be shared by multiple processes and thus represent multiple nodes.

Nodes (or really the programs on which they are based) reside on specific processors. If, for example, the system contains two processors of Type X, then it will probably be the case that a node for one Type X processor can be run on the other. While it may be the case that a node which can run on a Type X can not run on a Type Y processor, it is possible to have another node for the Type Y processor which performs the same function as the Type X node. Hence, the ARC will support both node transferability and, for selected functions, node function transferability between processors. The implication of this is that nodes (i.e., their associated programs) must have unique identifiers and a list of processors on which the node may execute. Likewise, nodes should be grouped by function, such that a node which performs a particular function on one processor can be substituted for another node which performs the same function on a processor that for reasons of processor failure or workload is not available.

An important tenet of the ARC advanced design is that nodes are viewed as having a general cyclic structure. The basic node cycle includes reading in a specific amount (or range) of data from a link, processing the data, and writing out a specific amount (or range) of data as results of the processing to another link. Each cycle of node operation consumes input data from one or more links and produces output data on one or more links. Similarly, each cycle of node operation takes one step toward emptying the input links and filling the output links. If a node is permitted to run until it either empties one or more of its input links or fills one or more of its output links, then the number of cycles of processing that a node is permitted is dependent upon the size of the smaller of the input and output links. Smaller links fill up faster than larger links, hence, the node runs a shorter time to fill the smaller link than it does to fill the larger, assuming that its input consumption and output production remains fairly constant for a cycle. From this it can be seen that node execution can be scheduled by only running a node until the required link empty or link full events occur and then setting the size of the links to permit the desired amount of node processing cycles. This will be discussed in more detail when the scheduler is described.

As mentioned previously, a node receives data on links and outputs results on links. This is accomplished through the Metaexec link read and link write system calls. If a node desires to store data on disk, output data to a display, or input data from a terminal, it must send or receive, as appropriate, data to the spool (disk storage), display, or terminal systems nodes. The regular user node can only receive and send data through the link read or write mechanism.

2.2.1.2 Run Time Linker

The previous discussion has introduced the node and the node/link interaction in the scheduling decision. This section will look at the link in the context of the intra-processor link function which was termed the Run Time Linker (RTL).

The RTL exists to implement the Read Link and Write Link system calls. It performs the task of moving data from one node to another within a processor. The RTL is intended to be an interprocess communications mechanism, not a storage system. Hence, it implements links in memory, not on secondary storage. Preliminary analysis indicates that the access overhead for secondary storage of links (seek plus latency) is so severe that in general the real time processing requirements of the ARC can not afford this magnitude of overhead for links. If secondary storage of data is desired, the experimenter must specify the transfer of data to a spool node.

2.2.1.2.1 Sublink Design. The link is a logical concept of internode data movement which is implemented through the use of sublinks (a type of physical buffer). The sublink, illustrated in Figure 2-3, is the physical component of the link. It is the sublink, or more correctly, a pointer to the sublink which is transferred from the source node (the node that is writing into the link) to the destination node (the node that is reading from the link).

In order to provide some organization to the discussion, the sublink design will be viewed from the perspective of sublink size, sublink allocation to links, and sublink utilization. Each of these issues will be considered in turn.

LOGICAL LINKS ARE COMPOSED OF
PHYSICAL SUBLINKS

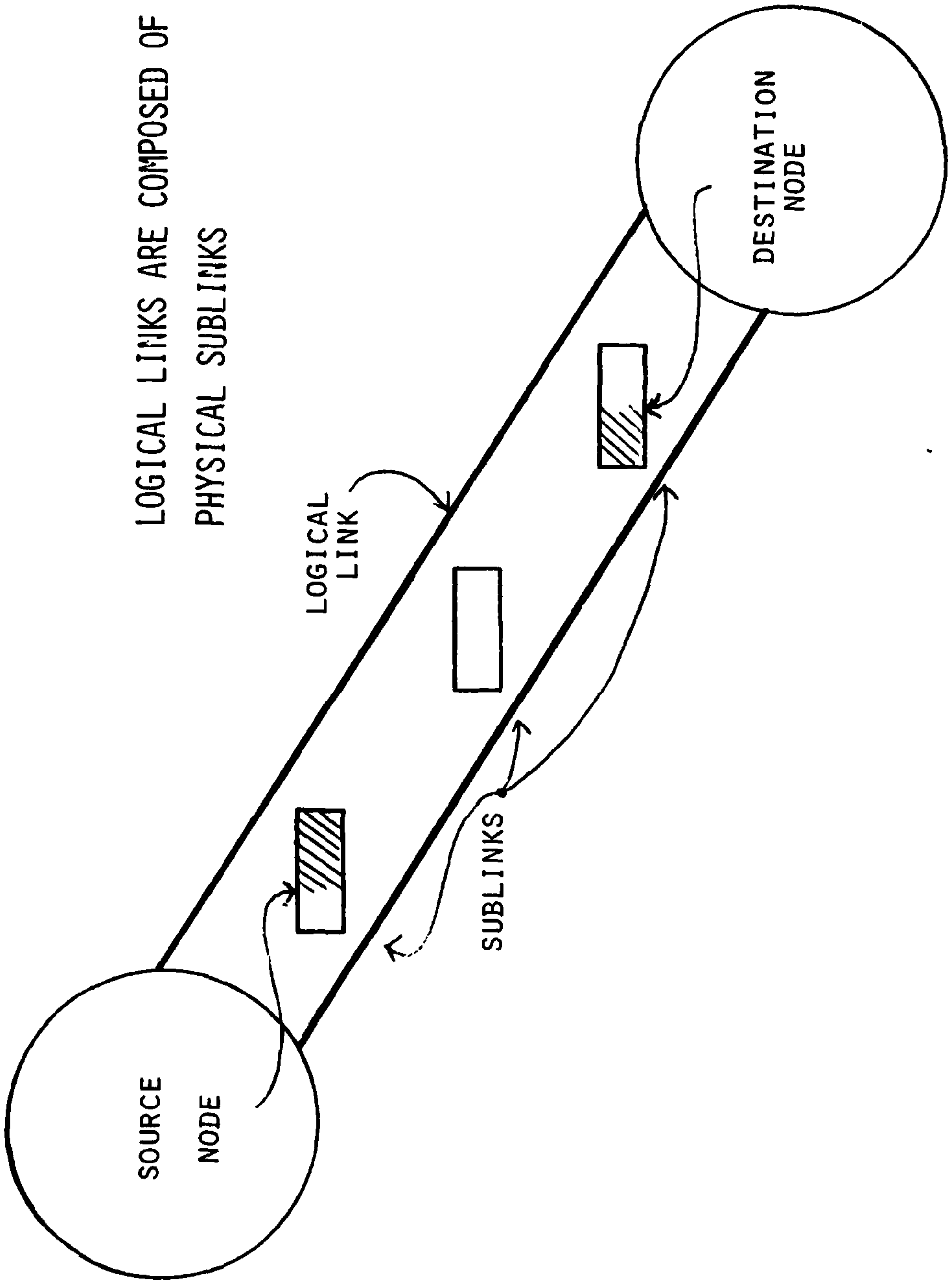


FIGURE 2-3 SUBLINKS

Sublink Size

Three possible approaches to the choice of sublink size were considered in the design. The first choice was to permit only a single size sublink for the entire system. This approach was deemed unacceptable because of the potentially large differences in node cycle output and input requirements and the close correlation between node scheduling and link filling and emptying (which translates into sublink filling and emptying). A sublink size more proportional to node consumption and production requirements is desired.

A second approach to sublink size is to permit sublinks to be of any arbitrary size as determined by node consumption and production requirements. If a common sublink pool area were to be used, this approach would cause fragmentation of the pool memory area and necessitate garbage collection. In addition, if source node production is of a different size than destination node consumption, either a sublink whose size is the least common multiple of the production and consumption amounts must be used, or else a size mismatch exists for either the source or destination nodes. The least common multiplier is deemed unacceptable because its size may be much larger than that which is desired for the desired node activation duration. If, rather than using a common buffer pool, a fixed allocation of sublinks to links is utilized, the arbitrary sized sublink approach still has the overhead of each sublink having to specify its arbitrary size. Because of the overhead of the arbitrary sublink size approach and the fact that it precludes an efficient common pool arrangement, this second approach was also rejected.

The final approach considered and the one ultimately selected was to permit the system to support a number of different sized, but fixed length, sublinks. The sublink size selected for use by a particular link should be greater than the link production and consumption anticipated for the desired amount of source and destination node processing. Since this size will probably not be the least common multiplier, especially if there exists a variance to the production and consumption data amounts, there will exist a mismatch between sublink size and production data and/or consumption data amounts. Such a mismatch has the effect of either causing unused sublink capacity since the next production would overflow the sublink, or requiring a production or consumption to bridge two sublinks.

The choice between the unused capacity and sublink bridging approach is dictated by the potential need to support the situation in which node production into a link differs from node consumption from the link. This capability could lead to the situation in which for a given sublink, the consumption requirements of the destination node exceed the remaining data in the sublink. Such a situation is a virtual certainty if production amounts are exceeded by consumption amounts and the sublink is not a least common multiplier of these amounts. For this reason, the design must support the sublink bridging approach.

Sublink Allocation

Sublinks can either be permanently allocated to links, or they can be borrowed from and returned to a common sublink pool when they are no longer required by the link. The trade-off between the two approaches is more memory to support permanent allocation versus the possibility of experiment deadlock if the sublink pool is depleted. While this subject requires a more analytic study prior to making a final decision, it is tentatively recommended that sublinks be permanently allocated to the links. Hand simulations on simple models have indicated that sublink allocations significantly below that required by permanent allocation can only be obtained for experiments which are very deterministic and which are susceptible to the determination of an optimal, deterministic, global scheduling algorithm. The potential complexity and nondeterminism of ARC experiments makes it somewhat unlikely that such an optimal global scheduling algorithm can be developed, hence the recommendation in the scheduling section to follow of less than global scheduling with the resulting implication of high sublink utilization. However, it should be stressed that a final determination must await a more quantitative analysis of the ARC processing environment. If such an analysis is not feasible, then it is recommended that the RTL be capable of handling both permanent allocation and temporary allocation from a common buffer pool. With this capability, the system performance can be evaluated and the allocation scheme changed if deadlock proves to be a problem.

Sublink Utilization

The question of how sublinks are to be used to support the concept of a link is dependent upon the nature of the nodes utilizing the link and the nature of the link itself.

Nodes associated with a link may either be non-simultaneous or simultaneous. For the purposes of this report, non-simultaneous nodes are defined as those nodes associated with a single link which share a single processing unit (e.g., processor, I/O device, etc.) and which therefore can not both be active at the same time. Simultaneous nodes are nodes which share a link and which can both be active at the same time (e.g., one node may operate on the processor while another node is supporting a DMA I/O transfer).

Non-simultaneous nodes require at least three sublinks to insure that no deadlocks can occur. Under a worst case scenario, the destination node could have emptied most of its sublink, but its next consumption would require more data than currently available. Under the sublink utilization approach recommended in this design, the sublink could not be emptied until a second full sublink, containing the remaining piece of the consumption, were provided. This must be provided by the associated source node. Again under the worst case scenario, assume that the source node fills most of its sublink but that its next production would overflow the sublink. In this case, no additional production can be provided until an empty sublink is provided. This third sublink is required to break the deadlock which provides a full sublink to the destination node, thus permitting it to make its next consumption. The buffer requirements can be reduced to two if the source sublink can be shipped to the destination in a partially filled condition. Note, however, that the sublink requirements can not be reduced to one without providing the node with the ability to make a partial consumption -- just enough to empty the remaining data from the output sublink. This capability implies internal buffering within the node and the ability to differentiate between complete and partial data consumptions. This makes the link mechanism too visible to the nodes and will not be adopted in this design. The choice between the two sublink and three sublink approach will be left to the design stage.

The sublink requirements for simultaneous nodes differs from those for non-simultaneous nodes only insofar as the simultaneous nature of the nodes may make it desirable to double buffer -- that is to insure that source sublinks and destination sublinks can be operated on simultaneously. Since the three sublink approach provides this at all times and the two sublink approach provides this some of the time, it follows that the triple sublink approach should be even more seriously considered for links involving simultaneous nodes than for links involving non-simultaneous nodes.

The next aspect of sublink utilization to be considered is the requirement to support merging links, as defined in the LINGO report. A merging link consists of two source nodes feeding through a "Y" into a single link with one destination node. At least two approaches are apparent to support this concept, with the latter appearing to more closely support the spirit of the merging "OR" link. The first approach would have both source nodes share a common sublink. The second approach permits each source node to have its own sublink and each sublink is moved to the same destination node when it is full. In this latter case, it is necessary to insure that the design maintains the correct order since consumptions from links from either source may bridge two sublinks. With proper bookkeeping, however, the bridging can be properly handled, and this latter approach provides more of the flavor of a true "OR" of links than does the first approach. While the latter approach is preferred, both should be investigated during the implementation phase since overriding advantages of one over the other may appear at that stage.

The final concern of this section on sublink utilization is the support that must be provided to differentiate Running Links from Memory Links. The primary difference is that a sublink for a Memory Link is not deallocated from a destination node after it is emptied. This last value is maintained until a new consumption arrives on the next sublink. Since consumptions can bridge two sublinks, this means that a memory link can tie up two sublinks at the destination and two (or one, if partially full sublinks are permitted) more at the source requiring a maximum of four sublinks rather than just the three required for Running Links.

2.2.1.2.2 Link Event Machine. The heart of the RTL is the Link Event Machine which is a table driven function that allocates, moves, and deallocates sublinks to/from nodes and apprises the scheduler of Link Events. The Link Events are used by the scheduler to determine when to activate and deactivate nodes.

The basis for the Link Event Machine (LEM) actions are the sublink events of full, empty, almost full, and almost empty. While the first two events are self explanatory, the latter two require some explanation. An almost full sublink event occurs when a sublink reaches the point that it can no longer hold an entire node cycle production. An almost empty sublink event occurs when the sublink can not support another complete node cycle consumption (i.e., the next consumption must bridge two sublinks).

Figure 2-4 is an example of a possible Link Event Machine which uses the triple sublink approach discussed previously. This LEM is for a standard Running Link and does not include the slight additional complications of the Merging or Memory Link. It is, however, illustrative of how the LEM portion of the Run Time Linker will operate. The events along the left column are sublink events. The horizontal axis indicates whether sublink is attached to a source node or a destination node. In the case of a sublink filled event for a sublink attached to a source, the following actions are performed by the LEM:

FIGURE 2-4 LINK EVENT MACHINE

EVENT	ATTACHED TO SOURCE	ATTACHED TO DESTINATION
FILLED	ATTACH TO DESTINATION SOURCE OUTPUT LINK FULL EVENT DESTINATION INPUT LINK FULL EVENT	———

EMPTY	———	DETACH SUBLINK FROM DESTINATION
		IF SECONDARY THEN MAKE PRIMARY
		IF NO 2ND - THEN SEND DESTINATION LINK EMPTY EVENT
		ATTACH SUBLINK TO SOURCE AND SEND SOURCE OUTPUT LINK EMPTY EVENT

ALMOST FILLED	ATTACH SECONDARY SUBLINK	———
ALMOST EMPTY	———	SEND DESTINATION LINK EMPTY EVENT

1) The LEM detaches the sublink from the source node and attaches it to the destination node.

2) The LEM forwards to the scheduler a source output link full event indicating that the source's output link can currently receive no more input. It also forwards to the scheduler a Destination Input Link Full event which indicates that the destination's node can now read data from the link.

In the case of an Almost Filled event for a source sublink, the only action is to attach a secondary sublink to support the anticipated next node production which will bridge both sublinks. This next production will also cause a sublink filled event which will execute the previously described steps.

In the case of a sublink empty event for a destination sublink, the actions are as follows:

1) Detach the sublink from the destination node.

2) If a secondary sublink exists, move it up to the primary position.

3) If no secondary sublink exists, then send a destination input link empty event to the scheduler.

4) Attach sublinks to source and send source output link empty event.

If a sublink pool were utilized instead of permanent sublink allocation, the attachment of empty sublinks to source node outputs would have to be coordinated with the node scheduler. The LEM would also be changed to show empty destination node input sublinks going to the sublink pool, rather than the source node.

2.2.1.3 Scheduler

The scheduler is responsible for initiating the running of nodes (i.e., processes) on the processor. The running of a node (its activation) as well as its deactivation are dependent upon the activation and deactivation requirements specified by the language. Such requirements are specified in terms of the availability of input link data (link full event) and the availability of output link capacity (link empty event). The scheduler must be able to accept the node activation criteria as a parameter, since it may vary for different languages or for different releases of the same language. A node activation criteria suggested in the LINGO language is that a node must have data available on all of its input links and output space available on all of its output links in order to be activated. Under this criteria, the node would not be initially activated until the scheduler received link full events on all input links and link empty events on all output links. These events would be sent to the scheduler by the Run Time Linker.

2.2.1.3.1 Global vs Micro Scheduling. The fact that according to the activation criteria, a node is capable of being given control of the CPU or run, does not mean that it necessarily is given control. The question of when a node is actually run involves the question of the basic scheduling philosophy to be supported in the design. The two scheduling philosophies addressed in this design are termed the micro scheduling approach and the global scheduling approach. Since the latter is just an optimization of the former, the micro approach will be presented first.

Under micro scheduling, each node is treated as an independent process. As a node satisfies its activation criteria it would be placed on the ready to run queue and given control of the CPU when it reached to the top of the queue. Priorities might even be assigned to nodes in order to give them priority in reaching the top of the queue and thus provide a means of favoring certain nodes or strings of interconnected nodes in order to, for example, insure that particular hardware resources are more fully utilized than other resources.

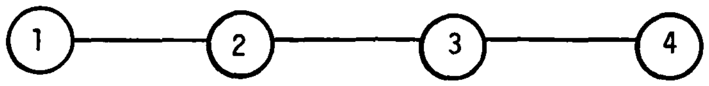
The alternative to micro scheduling is global scheduling which recognizes that nodes are not independent processes and that an experiment specifies a particular data flow and associated node activation sequence. Figure 2-5 illustrates the differences between the two scheduling concepts. Assume that each node is joined by only one sublink which for simplicity is precisely the correct size for node consumption and production. Since there is only one link, a node can not fill a link until the destination node empties it. Likewise a link can not be emptied until filling is complete. Also assume that all nodes are on the same processor. Under micro scheculing, the sequence of node activations illustrated in Figure 2-5 might be specified by the scheduler. At the end of the last node activation, the micro approach has managed to output one production (caused by the activation of node 4), but has left three link full. In contrast global scheduling has output two productions and has left only one link full. In steady state, both scheduling mechanisms will, however, average the same number of output productions per unit time.

While global scheduling offers the optimal scheduling sequence in this simple example, it suffers from a number of problems in the general ARC environment. The first of these is its ability to handle multiple experiments. If multiple relatively independent node/link strings are to be concurrently scheduled, then a global scheduler would need to essentially time slice between the separate optimal scheduling sequences for each string of nodes. This requires that the time slices be properly determined to insure that both experiments adequately meet their real time requirements. Secondly, the global scheduling approach assumes that an optimal scheduling sequence can be determined. For a complex experiment involving multiple nodes on different processors, with asynchronous I/O processing, such an optimal sequence may be difficult to determine. For these reasons, the design will support only the micro scheduling approach.

While the micro scheduling approach is somewhat more complex than the global scheduler, since it must monitor link events and determine for each event whether a node has satisfied its activation criteria, it will provide the same

FIGURE 2-5 MICRO AND MACRO SCHEDULING

Experiment

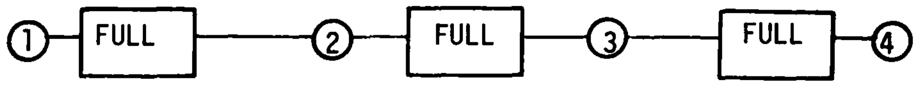


Micro Scheduling

-Node Activation Sequence

1, 2, 1, 3, 2, 1, 4, 3, 2, 1

-Final Link Status

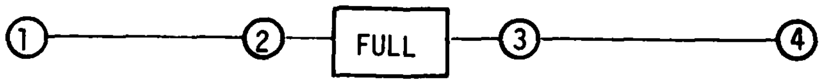


Macro Scheduling

-Node Activation Sequence

1, 2, 3, 4, 1, 2, 3, 4, 1, 2

-Final Link Status



output production data rate as global scheduling. It also has the added advantage of automatically compensating for multiple experiments since each node of each experiment is treated as an independently scheduled process.

Under micro scheduling, the following node states will be supported:

1) Running

2) Ready to run (node has all links)

3) Blocked (node has input links but no output links)

4) Inactive (node waiting for input links)

5) Wait (e.g., spool node waiting for I/O)

2.2.1.3.2 Micro Scheduling and Real Time Processing. While the ARC is a real time processing facility, it is not real time in the sense that any arbitrary experiment node must support real time, asynchronous events. In fact, it is the case that the source nodes (which collect the raw data) must collect real time data, but at a well defined, synchronous rate. Since the source nodes feed data to all of the other nodes of an experiment, then it is possible to decouple all subsequent nodes from any variance in data rates that may be introduced into the data stream (e.g., through the transmission mechanism). This is accomplished by requiring that all data be spooled to a sufficiently large secondary storage device. This is called for in the design. All remote site data will be spooled to disk upon arrival at the central site. This means that only the remote processing nodes and the central site communications input function must support real time processing. In order to insure the necessary real time response, the schedules will give higher priority to these nodes and functions than it will to those which have been decoupled from real time by the spool.

A second concern that must be addressed by the micro scheduler approach is that of insuring that special purpose processors (such as the CHI in the present ARC configuration) are not left idle because no data has been written on its input link. Unfortunately, because of the lock step sequencing of experiment nodes, those nodes operating on a CHI, for example, can only process data at the rate of the slowest node that is upstream of it (i.e., between it and the source nodes). It is, however, possible to assign a higher priority to all of those nodes which are upstream of the CHI node. This has the effect of insuring high utilization of the CHI, but it will also limit the processing capability of those nodes not assigned a high priority. This may result in the under-utilization of another processor fed by these low priority nodes.

In order to provide the director with the ability to adjust the performance of experiments and the utilization of processors, the scheduler will provide the capability of activating nodes from the ready queue according to priority. Such a priority scheme will, however, have to be somewhat dynamic in order to insure that all nodes eventually receive the necessary amount of service from the CPU.

A final aspect of the scheduling which has an impact upon system utilization and real time performance is the question of the metering of the data flow. If an experiment is running, using data provided by the real time source nodes, then its processing rate is implicitly metered by the rate of actual data collection. If, however, an experiment is being run using data from spool node, then there are no implicit limitations upon experiment processing rate. All nodes in an experiment could theoretically run at a rate limited only by the transfer rate of the disk which supports the spool node. Under the micro scheduling approach, such an eventuality would take up a major portion of system resources and have an adverse effect upon the performance of other experiments in the system. In order to prevent such an eventuality, the spool nodes must contain metering mechanisms which in a sense simulate the sampling rate mechanism of the source nodes. One method of supporting this capability is to utilize the real time system clock, and periodically place the spool node in a wait state until an appropriate time out occurs.

2.2.1.4 Interprocessor Linker

The interprocessor linker (IL) supports interprocessor communication between NL processors (those that support the Node/Link concept). Because of the diversity of NL processors, this function may have to support satellite links, internal bus type links, modems, and other standard communications interfaces. The IL will use each of these interfaces and associated protocols as the basis for implementing the standard Node/Link protocol.

This standard NL protocol will take in sublinks from the Run Time Linker and send them to either the destination processor or a processor which acts as a relay to the destination processor. This relay function will also be supported by each IL. The relay process will forward all non-destination messages on to either the proper destination or the next relay.

The IL function could potentially introduce a queue buildup problem into the data flow. Such a queue problem has been previously avoided because of the lock step nature of the node scheduling within a single processor. With the IL function, however, a sublink on a source processor could be emptied by the IL function prior to the consumption of this sublink by the destination node on the destination processor. If appropriate measures are not taken, the source node could be rescheduled to fill up one or more additional sublinks prior to the emptying of the first sublink by the destination node. The solution to this problem is to build in some coordination between the two IL functions involved in the transaction. The source IL must hold onto its input sublinks until it receives a Link Empty Event from the destination IL. The destination IL must be informed by the destination scheduler when a sublink empty event is received for the sublink of concern. This means that the IL function must have activation requirements of "any output link empty" transmitted to its scheduler. Upon activation by the scheduler upon a link empty event, the IL will transmit the link empty event and associated link identifier to the source IL. Upon receipt, the source IL will release the appropriate sublink which will cause the source scheduler to note a sublink empty (available) event for the affected node. If anything should happen during sublink transmission to cause the loss of the sublink, the IL, by retaining its input sublink, can retransmit the sublink after an appropriate time period.

Under this scheme, queue buildup is avoided on interprocessor links and the scheduling of nodes on different processors is properly synchronized. If an experimenter should desire to build up a queue of sublinks at either end of an interprocessor link, then a spool node can be defined on either or both processors.

2.2.1.5 Gateway

The Gateway performs a function similar to that of the Interprocessor Linker. The major difference is that it must interface processors which do not support the standard NL protocols or the standard ARC Nodes and Links.

Since the experimenter describes his experiment in terms of Nodes and Links even for that portion of an experiment which takes place on a non-NL processor, the responsibility falls to the Gateway to insure that the non-NL processor performs the desired functions. Depending upon the capability of the non-NL processor interfaced by the particular Gateway, this may include outright control of the non-NL-processor down to the level of commanding the swapping in of the necessary programs, setting of the necessary program parameters, and transferring the data that is to be acted upon to the processor. If the non-NL processor is sufficiently powerful, the Gateway may only have to forward the data to be processed along with a list of the processing that the non-NL processor is to perform.

Since the Gateway is responsible for insuring that the experimenter specified Nodes and Links are performed on the associated non-NL-processor in some manner other than with a Node/Link approach, it is necessary that the Gateway receive that portion of the local operations orders that specifies the functions that its non-NL-processor is to perform in the experiment. Likewise, the Gateway must handle all of the other functions that the Local Metaexec performs for the NL processor including system status and performance assessment, debugging, hardware status monitoring, and, in some cases, scheduling.

Gateways provide a convenient way to expand the functional capabilities of the ARC through a standard interface function. Such a function can be used to interface such diverse processing capabilities as a CHI parallel processing computer, the ILLIAC IV, or the ARPA net with particular ARPA net hosts. The Gateway's communication with other nodes in an experiment over the Run Time Linker provides a convenient interface protocol through which added capabilities can be easily added to the ARC as required without causing a major impact on the ARC software. Since the Gateway, although called a system function, has the attributes of a node, the experimenter can easily include the Gateway's non-NL-processing capability in the experiment as a single node whose internal functions are known only to the experimenter, or as a more complex processing environment in which the Gateway simulates the Node/Link capabilities of an NL-processor.

2.2.1.6 Node/Link State Function

The Node/Link State Function possesses and maintains the common tables used by the other Local Metaexec (LME) functions. At this point in the design, two tables have been identified and described. These are the Link Table and the Node Table.

Figure 2-6 illustrates the Link Table. This table provides basic link information and keeps track of the sublinks associated with each link. It is loaded by the local loader and used by the Run Time Linker. For each sublink, it indicates where it is allocated (source primary or secondary or destination primary or secondary) and the state of the sublink.

Figure 2-7 illustrates the Node Table which is used by the scheduler to activate and deactivate nodes. The table is self-explanatory except for node activation

FIGURE 2-6 LINK TABLE

LINK ID

SOURCE NODE ID

DESTINATION NODE ID

LINK SIZE

SUBLINKS

SUBLINK SIZE

	Sublink #1	Sublink #2	Sublink #3
Allocation (Source 1 or 2; Destination 1 or 2)			
State (Full, Almost Full, Empty, Almost Empty)			
Sublink Address			

FIGURE 2-7 NODE TABLE

NODE ID

PROGRAM ID

NODE MODIFIERS (Load time parameters which change the function of the Node)

NODE REQUIREMENTS (Memory, etc.)

NODE ACTIVATION LIST

NODE STATE

- RUNNING
- READY TO RUN (Node has all Links)
- BLOCKED (Node has input links but no output links)
- INACTIVE (Node waiting for input links)
- SWAPPED

NODE ADDRESS

NODE FLAGS

list and node flags. The node activation list is a list of the node's input and output links with their current state (available for input, available for output, or not available). This table is set by the scheduler when a link event is received. A link full event on an input node, for example, would set the entry for that link in the node activation list to available for input.

The node flag are special programming restrictions or conditions that apply to a node. The only restriction currently identified as necessary is a flag which indicates that a node's program should be locked into memory since it is known that it will be in almost constant use.

2.2.1.7 Local Control

The Local Control of the LME acts as the Metaexec interface between the LME modules and the CME functions. It receives commands and requests from the CME which are passed to the appropriate LME module, and it forwards responses from the modules back to the CME. The Local Control has the job of starting and stopping source nodes, on command from the CME control. This has the effect of starting and stopping experiments. It is also in charge of initialization of the processor LME and establishing the necessary connections to other processors in the ARC.

Messages to an LME module from the CME arrive at the Local Control module via the Run Time Linker. If the CME is on the same processor as the LME it is directly connected to the Run Time Linker, whereas if it is on a different processor, the connection is made through the Inter-Processor Linker. When a message arrives at the Local Control it must be examined to see which module it will be handled by, and then it is passed to that module. All module replies are prefixed with a header which indicates the source of message in terms of module and processor, and the CME destination function. These are passed to the Run Time Linker for delivery to the CME.

The local control executes commands from the CME Experiment Control function to start and stop source nodes. In terms of the ARC, starting a source node means having it begin to write data on an output link. A source node is a system node and may already be supplying data on links to other experiments, but when it begins to supply data on a new link, it effectively activates the new experiment. Thus by starting the source node, the experiment is started. Stopping a source node causes end-of-file tokens to be generated on the output links of source nodes, and as these propagate through the network, the experiment becomes inactive or stopped.

On initial startup of a processor, the Local Control establishes connections in the Run Time Linker between other LME modules and the CME. The module can also be used to activate backup CME functions if the processor supporting the normal CME becomes inoperative.

2.2.1.8 Local Loader

In order to install nodes and links for an experiment network, each processor must receive a set of Local Operation Orders describing the nodes and links it must support. These orders are received from the CME loader and are passed to the Local Loader. They are used to set up tables within the Node/Link State function. These tables represent the nodes and links to the LME and are used by the Scheduler and Run Time Linker to support the execution behavior of the node-link language.

When the loader receives a set of L-OPS it must examine them and verify that they can be handled by the processor. This verification includes making sure there are sufficient unused entries available in necessary tables, that all programs are accessible, and that the buffer requirements can be met. If the loader determines that the processor would not be able to support the experiment, it discontinues processing and informs the Central Loader of the problems. Otherwise the table values are set.

When all the tables have been set up, the experiment is considered ready to run, and the Local Loader notifies the CME loader. The experiment begins running either when it receives data on input links, or if it contains source nodes, when these are started by the Director.

When an experiment is not running it may be unloaded by a command from the CME loader. This causes the Local Loader to go through the tables of the Node/Link State function and remove all entries pertaining to the experiment. When all the information has been deleted from the tables, the Local Loader notifies the CME loader. Thus the Local Loader is responsible for overall management of the Node/Link State Tables.

2.2.1.9 System Status and Performance

The System Status and Performance (SSP) module monitors experiment status and system behavior, and responds to requests from the CME Experiment Monitoring and Tuning function. The SSP module makes all status information available to the CME for analysis. The SSP is also capable of causing performance monitor tokens to be inserted in data streams for data flow monitoring within the processor, and between different processors.

Specific monitoring that is performed by the SSP includes node processing times, node invocation rates, data input and output quantities, link utilization and system overhead. This information can be used by the CME for tuning and experiment optimization.

2.2.1.10 Hardware Status

The Hardware Status module monitors the state and utilization of the LME processor, and all devices connected to it. All the status information is available to the CME Machine Monitor which periodically polls each of the Hardware Status Modules to determine the overall ARC machine status.

The module monitors CPU and device utilization, core and buffer allocation, and the condition of the operating system. This information is passed to the CME on request where it is used to provide ARC system status and may also be used for failure analysis recommendations. The status module must also respond to a periodic status inquiry from the CME. The positive response is given if the hardware is operating normally, and either a negative response or no response if there are problems. If any of the critical modules of the LME are not performing properly then it will be impossible for the LME to respond positively to the inquiry. The CME will detect the problem condition and initiate checking procedures. If necessary, failure analysis will also be performed.

The Hardware Status module must also monitor the state of all devices attached to the processor. For non-NL processors, this monitoring is done through the Gateway function of the LME.

The entire monitoring system can be viewed as a hierarchical three-level system. At the top level is the CME monitor for the entire ARC. It receives all its information from each of the status modules in the LMEs, and provides overall system status. Each of the LME Hardware Status modules provide second level monitoring by determining the status of their processor, and the status of all attached non-NL processors. These will provide the third level monitoring information to the supporting LME.

2.2.1.11 Debugging

The LME must contain a debugging facility that provides convenient aids for the testing of new experiments. These aids would be usable by the experimenter at a single processor or could be coordinated between multiple processors by the CME Debugging function.

The debugging aids can be used in combination with the experiment monitoring function to provide the user with a large amount of information concerning the behavior of an experiment. Debugging aids include the ability to interactively control the starting and stopping of nodes, examination and modification of link values, and data flow tracing mechanisms. The debugging facilities can be coordinated between different processors, so that the inter-computer networking of an experiment can be tested.

2.2.1.12 System and User Nodes

In addition to a CME and LME, an operational ARC facility will require some standard system nodes to provide widely used functions required by most experimenters. System nodes to be provided include Source Nodes, Spool Nodes, Display Nodes, and Terminal Nodes.

The Source Nodes will control the original data capture mechanisms of the ARC. Through control parameters, the Director will be able to set the sampling rate, select the source, and control the distribution of source output to competing experiments. In addition, the Source Nodes will provide the necessary multiplexing and time tagging functions.

The Spool Node will perform the data management function for ARC data. The Spool Node will be able to store and retrieve data according to the time recorded, site, beam, sensor, and previous processing.

The Display and Terminal Nodes will provide the necessary data processing required to prepare data for display or accept data from the terminals.

The user nodes will be specified or provided by the experimenter. They may, however, contain one or more standard node functions which facilitate the node's interface to the experiment network. The primary standard node functions identified thus far are functions to handle time tags and functions to handle multiplexed data. It is anticipated that both user nodes and system nodes (such as the Spool Node and Source Node) may use these functions.

2.2.2 CENTRAL METAEXEC

The Central Metaexec (CME) contains a set of functions that control all the overall operations pertaining to the ARC. Some of these functions are oriented toward the Director's role of managing the general actions of the ARC while other functions are oriented toward the user and his experiment. Several of the functions are shared by both the Director and users. The Director oriented functions include Experiment Loading, Experiment Control, Experiment Monitoring and Tuning, Machine Monitoring, and Failure Analysis and Recommendations. User oriented functions are Experiment Monitoring, and Debugging. Each of these functions, along with the Director Interface and User Interface will be described in further detail in this section. The relationship between these functions is shown in Figure 2-8.

The CME design is not tailored to any specific processor in the ARC. It can be run on any Node-Link (NL) processor, and its design is such that it is possible to distribute the CME functions on several processors.

The CME has a dual relationship to Local Metaexecs of the Node-Link processors. The CME is the overall control and monitor of all the LMEs. It provides Local Operation Orders (L-Ops) and receives status information for each processor. Thus the CME acts as a centralized control of the entire ARC system. However, the CME is designed to reside within the confines of one or more LMEs, and thus is dependent on its host LME for invocation and communication. This relationship can be characterized in the following manner. The LMEs provide general computer support for programs that may be used for data processing or ARC control. The support includes general communications between programs on the same or different processors, CPU cycles, and peripheral access. The programs that comprise the CME specify which data processing programs should be used and how they should be interconnected. The CME produces these specifications based on the Global Operation Orders (G-Ops) received from the LINGO language processor. Thus, the LME's support

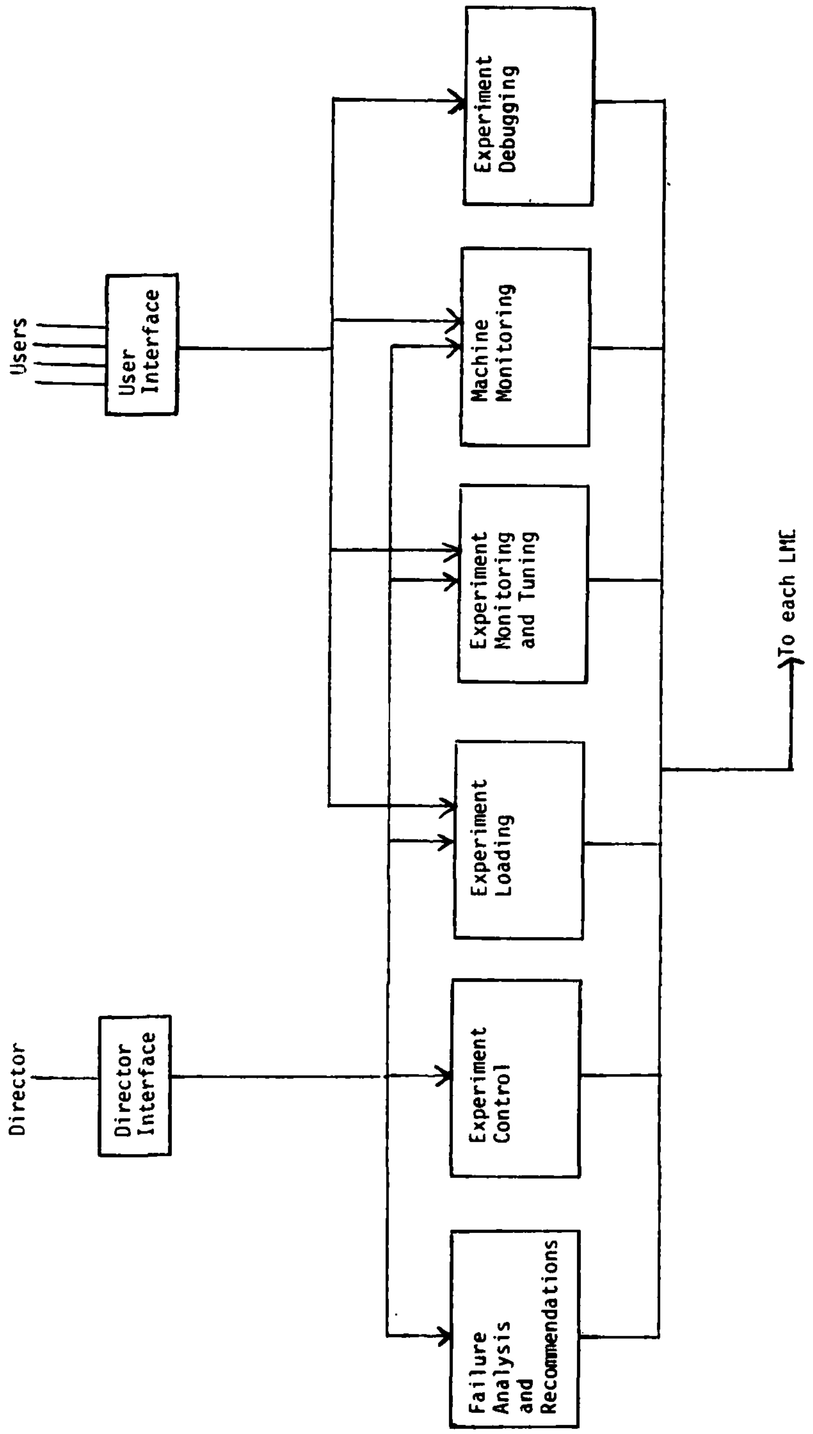


FIGURE 2-8 CENTRAL METAEXEC FUNCTIONS

the CME and data processing in a similar manner. However, the CME acts as the overall management for the data processing, and exercises this control through commands to the LMEs.

The CME functions are implemented very much like the system and user nodes. Like nodes, the functions are invoked by the LME scheduler, but they need not strictly follow the node invocation criteria. It is necessary for the CME to have connections to each of the LMEs so commands and status information can be exchanged. These connections are from the CME to the Local Control module in each of the LMEs. Each connection goes through the Run Time Linker of the CME's processor and then either through a system link to the host LME, or through the Interprocessor Linker to LMEs on other processors. These connections are set up dynamically as part of ARC configuration initialization.

The division of the CME into distinct functions allows these functions to be distributed across several processors, if necessary. When the functions are separated in this way, it is necessary to establish connections between them. This is done analogously to interprocessor links between nodes. When multiple CME functions reside on the same processor, they may be combined so that interfunction communication is more efficient.

2.2.2.1 Director Interface

The Director Interface (DI) is the function that provides two-way communication between the ARC and the Director. The DI provides updating displays representing the status of ARC hardware processors and devices, LME operation, and experiment states. It also notifies the Director of any exceptional events that occur and require Director intervention. The DI interprets requests and commands and passes them to the appropriate CME function for action. All other CME Director oriented functions are effectively accessed through the DI. These include all the functions identified in the previous section.

The physical interface supported by the DI is an alpha-numeric character terminal. Multiple terminals may be used for simultaneous presentation of displays, and so that the Director responsibilities can be divided between personnel. The DI will use a terminal command language that allows easy switching between functions, and a concise command format.

2.2.2.2 User Interface

The User Interface (UI) provides two way communication between the ARC and each ARC user that is similar in its function to the DI. It provides the user access to a different set of functions that includes a debugging function. The physical interface and command language is similar to the DI, although not as many capabilities are provided. The UI supports user terminals at any connected location, including over the ARPANET.

2.2.2.3 Experiment Loading

At user request, and under director control, an experiment can be loaded into the ARC through the Experiment Loading function. Before an experiment is loaded it is assumed that its LINGO description has been passed through the LINGO processor which has generated a set of Global Operation Orders and a list of experiment requirements. These will be stored on a medium that is accessible to the Loader. A user wishing to run an experiment signs on to the ARC through the User Interface. He is then able to check the status and the current utilization at the ARC to determine if it is a reasonable time to attempt the experiment. If the system appears capable of supporting the experiment at an acceptable processing level, the user would send a message to Director requesting that the experiment be run.

The Director would use the Experiment Loader function to first analyze the requirements for the experiment to determine the effects of introducing a new experiment on the current experiment mix. The loader does this by comparing the experiment provided requirements with the current system loads.

Typical requirements that would be checked include processor utilization, spool and storage needs, remote site requirements, and use of exclusive access devices, such as special display hardware. The impact analysis of a new experiment would be performed by comparing each of the resource requirements to the current utilization. If the resulting resource utilization levels are acceptable to the Director, then the loading of the experiment could proceed. The introduction of a new experiment may reduce the performance of an existing experiment, but these effects should be identifiable by the loader analysis. It should also be noted that the resource requirements should give both best and worst case estimates, as well as a most likely or average value. This type of information will aid the Director in making a decision that is more representative of the actual and theoretical resource utilization loads for the system.

After the resource utilization analysis is completed, the Experiment Loader would enter the decomposition phase. In this phase, the Global Operation Orders are separated into Local Operation Orders for each processor. The G-Ops produced by the LINGO processor are actually made up of a group of L-Ops. An example of the form of the L-Ops is given in Figure 2-9. In decomposing them, the loader checks for completeness of the orders, and fills in any missing load time parameters by queries either to the experimenter or Director. Missing load time parameters could include selection of processors when a choice is available, or specification of which terminals will be used by the experimenter. When the L-Ops are complete they are sent to Local Loader of each processor. The Local Loader incorporates the operation orders into the Node/Link State Function of the LME, and responds back to the CME loader when the incorporation is complete or if errors occur. When positive acknowledgement is received from all required processors, the experiment is considered to be loaded, and the Director is notified. To start execution of the experiment the Experiment Control function is used.

FIGURE 2-9 TYPICAL L-OP FORMAT

NODE TABLE

Node 1

Node ID

Program type

Processor type (if generic)

Resource requirements

Inputs

Outputs

Node 2

⋮

Node n

LINK TABLE

Link 1

Link ID

Source node ID

Destination node ID

Suplink size

Type of link (memory or running, inter or intra processor)

Initial values if memory link

Link 2

⋮

Link n

A final operation that is performed by the Experiment Loader is unloading of experiments. When a loaded network is not running, the Director may remove it from the system by unloading it. This action is performed through the loader by the sending of signals to each LME loader. These signals result in the purging of table information associated with the experiment. When all the experiment relevant information is removed from the LME, it sends a notification back to the CME. When all notifications have been received, the Director is informed.

The Experiment Loader function thus consists of four phases. The first phase is used by the Director to determine the suitability of running a requested experiment. This phase is also available to the experimenter so that he can make his own determination of whether the experiment should be run. The second phase includes separation and checking of the L-Ops. In the third phase the L-Ops are distributed, and when all have been received, the experiment is ready to run. The final unloading phase allows for the freeing of resources that may be allocated for experiments.

2.2.2.4 Environment Control

The Experiment Control function is used to manage all loaded experiments. Through this function, the Director can start and stop experiments, and set parameters for them.

Starting of an experiment can only occur after the experiment is loaded and acknowledgement has been received from each processor. To start an experiment, it is only necessary to start each of the source nodes of the experiment. This is done by the Experiment Control sending a signal to each processor that contains a source node for the experiment. The signal goes to the Local Control of the LME, and causes the source nodes to begin to produce data on their output links. As links are filled, other nodes become activated, and the data percolates through the network.

Once the start signals have been sent by the Experiment Control function, the experiment is almost exclusively handled by the Local Control of the LME. The CME control is not necessary for the moment to moment operations of the experiment. However, if part of the network is shared between multiple experiments, any parameter changes to these nodes would impact the different users. These parameters might include things such as sampling rate, beam forming, or FFT size. Control of these parameters is exercised by the Director through the Experiment Control function. The CME provides information about the effects of the parameter change in terms of which users are using data that is derived from the node, and what the impact on the processing requirement would be. When the parameter is to be changed, a signal with the new value is sent to the LME that controls the node. At the Local Control the parameter change is handled in the same manner as a local parameter change.

Stopping an experiment is analogous to starting. A signal is sent to all source nodes indicating the data stream should be turned off. This causes end-of-file tokens to be placed on all the data links, and these tokens propagate through the network until all nodes have become inactive. The network is then in a similar state as the one it was in before it started. It could subsequently be either unloaded or restarted by the Director.

2.2.2.5 Experiment Monitoring and Tuning

A very important aspect of the CME is the ability to monitor the performance of an experiment, and make adjustments in system parameters and tables to optimize its behavior. These operations are performed by the Experiment Monitoring and Tuning function.

Experiment monitoring is chiefly done by receiving experiment status information from each of the processors and combining this information to provide an overall picture of the experiment behavior. The type of information would include activity levels of nodes and links, comparisons to predicted activity levels, and indications of how performance could be increased.

One method that will be used to monitor data flow will be the injection of special system monitoring tokens into the data stream. These tokens will be traceable through the system by the local monitoring functions and their progress will be reported to the CME monitor function. The use of monitor tokens is analogous to using a colored dye in fluid system. At various points within the system the arrival time of the dye can be monitored to provide performance information and detection of bottlenecks.

Information about network activity levels and performance can be used by the tuning function to provide suggestions of actions that would improve the experiment behavior. This function could analyze congestion, and propose remedies that the Director could dynamically implement during the experiment. These remedies could include changes in link size to allow nodes to remain active longer, or change in node priority to increase node service. The control tuning of the system could initially be very primitive, but would be improved and developed as the understanding of the ARC execution behavior broadened.

2.2.2.6 Failure Analysis and Recommendations

When an ARC component fails it is important that not all operations on the system be halted. In the event of a failure, the Failure Analysis and Recommendations (FAR) function will provide information to the Director on the extent of the failure, and alternative responses that may be taken.

There are two situations that must be presented to discuss this function. The simpler case is when the processor supporting the FAR and the failing processor are distinct. In the other case the FAR and possibly other CME functions are being supported on the failing processor, and therefore it is necessary to regain the central control before the failure analysis can proceed.

Whenever component failure occurs, the first problem is identification of the inoperative hardware. This is done by the Machine Monitoring function that is described in the next section. With this information, and with experiment configuration information obtained from the experiment monitor, the FAR can locate which nodes of running experiments are influenced by the failure. The FAR can then make recommendations for reconfiguring the nodes on other available processors. The FAR will use node descriptions and resource requirements to determine which nodes can be relocated on which processors. The recommendations may also be based on the Director's decisions as to which parts of experiments may be curtailed to free resources. Thus the FAR plays an interactive role with the Director to determine how essential ARC operations can be maintained. As alternative plans are formulated, new Local Operation Orders are generated, and these are passed to the Experiment Loader function to be incorporated into the LMEs.

As the failure analysis is being performed, any nodes on the non-functioning processor would be unable to run, and this would cause preceding nodes in the network to become blocked. This condition would eventually back up to either a spool, where incoming data would be stored for a period of time, or back to a source where data might be lost. When the corrective actions had been taken, and the replacement nodes started, the backed up data would begin to flow through the system. Under some conditions it would be possible for machine failure to occur, analysis and reconfiguration to take place, and processing to be restarted with minimal interruption to the experiment.

When a processor containing critical functions of the CME fails, the prognosis is more difficult. Other parts of the ARC will continue operation until nodes become blocked. In the meantime the Director would activate a backup version of the CME functions on another processor, and use this version to reconfigure the nodes on the system. When the reconfiguration is completed the backup CME would be used to bootstrap a more complete CME.

Like experiment tuning, the FAR functions could initially be rather simple, and grow in complexity as the ARC develops. The FAR could include some default contingency plans for some failures such as automatic switching to a backup processor and maintaining the data archive. The FAR is designed to allow for additions that are appropriate for the changing nature of the ARC.

2.2.2.7 Machine Monitoring

The Machine Monitoring function provides the Director and users with information on state of all the hardware in ARC system. It also has the function of determining machine failures, and aiding in failure analysis.

ARC System monitoring is performed by the Machine Monitoring function by periodically polling the Hardware Status module of each LME. The module returns information about the state of the machine and its peripherals. If an LME does not respond properly to the polling then the Machine Monitor instigates further checks to determine the hardware status of that LME's processor. The monitor notifies the Director if it determines that a processor is not operating correctly or if a processor reports that a peripheral device is down. The Director could then use the Failure Analysis and Recommendation function to derive an alternate plan.

In providing routine status information to the Director and to users, the monitor summarizes the received information, and makes detailed information available on request. The type of information available could include CPU utilization, disk utilization, core allocation, process activity and device allocation.

2.2.2.8 Experiment Debugging

The Experiment Debugging function provides the ARC user with a system wide capability for debugging new experiments. As with several of the other CME functions, the debugging functions work with individual debugging modules on each of the LMEs.

The types of testing features provided by the ARC include the ability to monitor traffic flow tokens placed in the system, running partial experiments on live or archived data, interactive checkout of experimental nodes, and examination of data flowing on links.

Because of the uniqueness of the ARC System, the only environment that is suitable for experiment testing is the ARC itself. It is therefore necessary that these comprehensive debugging functions be included in the ARC Metaexec.

3.0 REQUIREMENTS ALLOCATION SUMMARY

This chapter shall identify the allocation of requirements to the system software. For this report and design stage, the objectives given in Section 1 will be discussed in terms of the software design presented in Section 2.

3.1 LINGO SUPPORT

The LINGO language is supported by the Metaexec through both the CME and the LME. The CME causes L-Ops to be sent to each processor and provides the capability of starting, monitoring and stopping experiments. The LMEs support the actual execution of running experiments through the Scheduler, Run Time Linker and Node/Link State functions. These modules are designed to handle a variety of scheduling and invocation methods implicit in the LINGO network semantics.

3.2 REAL TIME PROCESSING

The Scheduler in each of the CMEs is aware of the timing constraints for each node and can use the scheduling discipline that is the most appropriate. Nodes that are running off spooled data can be scheduled in an event driven fashion, while nodes driven off live data can be scheduled on a priority basis. If too many nodes are introduced into the ARC, the system may become saturated and real time processing rates will not be maintained. The Experiment Monitoring function of the CME would indicate this event, and permit the Director to make modifications to the system to reduce loads, if experiment performance became unacceptable to users. This could involve favoring some experiments over others to maintain real time rates.

3.3 MULTIPLE COMPUTER SUPPORT

The top level Metaexec design is machine and configuration independent. The design is intended to support multiple computers, and a changing hardware configuration. This is accomplished by implementing the LME and a set of nodes on each NL processor. The processor can then be incorporated into the ARC in a generalized manner. The LINGO processor and the CME maintain tables that specify the capabilities of each processor, so that separation and monitoring of tasks among the processors can be performed. After the CME has distributed the experiment nodes and links on the different processors, the LME's on each processor are able to independently control the execution of their portion of each experiment. Because a standard protocol is used between each of the processors, inter-computer operations are simplified for the LMEs. The CME provides any centralized control or overall coordination that is necessary, and thus the Metaexec supports experiments on multiple computers.

3.4 DEGRADED OPERATION

Through the generalized CME - LME relationship, the Metaexec can support a degraded mode of operation during a partial system failure. When a failure occurs, the Failure Analysis and Recommendations function can determine the impact of the failure, and propose alternate network configurations to compensate for inoperative hardware. These can be dynamically incorporated into the system by the Loader, so that critical portions of experiments can continue with minimal interruption. If the CME processor fails, a backup CME can be activated on another processor and perform the same type of reconfiguration. With these mechanisms, extreme impacts of hardware failures can be minimized by the reorganization to support essential processing.

3.5 DATA CAPTURE

The Metaexec supports processing to capture all central site live data. This includes responding to interrupts in timely fashion, and maintaining spools and archives. These functions are handled by system nodes that are scheduled as input buffers are filled and output buffers are emptied. If the supporting processor fails, these nodes can be reestablished on other processors so that data loss is minimized.

3.6 SYSTEM EXPANDABILITY

The Metaexec is designed to support multiple processors in a general fashion, and is not geared to any specific configuration. This expansion of the system is easy to do, and potential expansion plans can be evaluated on a basis of added capabilities rather than Metaexec constraints and limitations. The expansion requires modification of tables in the LINGO processor and the CME, and implementation of a LME for the new processor. The system is also capable of supporting new Non-NL processors by the addition of new Gateway functions to connected LMEs. The use of standardized protocols between NL processors, and specialized protocols between Non-NL processors assures that expansion can occur with all kinds of devices and interfaces.

3.7 MULTIPLE EXPERIMENTS

The Metaexec is explicitly designed to support multiple independent and dependent experiments. New nodes and links can be established while experiments are being executed. These experiments can be started and stopped independently, and connected to each other through the Metaexec link mechanism. The monitoring and analysis functions can be used for determining the current system load and evaluating the impact of new experiments. This allows experiments to be started and stopped independently and with foreseeable results. It is intended that such changes to the experiment mix be accomplished with minimal interruption to the continuing experiments. The ability to predict resource utilization and experiment performance, as well as the resultant increase in reliability of the supporting software system leads to an ideal environment for dynamically changing experiment mixes.

4.0 TEST AND VERIFICATION

This section will be provided during the next stage of the development effort after the hardware configuration is finalized.

5.0 REFERENCES

[1] Benoit, J. W., et. al., "ARC User Interface and Metaexec Design Considerations," Mitre/Metrek, MTR-7511, April 1977.

[2] Rogers, J. W., and R. E. Murphy, "ARC Requirements Analysis Final Report," SDC, TM-SV-(L)-5805/001/01, June 1977.

[3] Uzgalis, R. C. and P. Eggert, "User Language Specification," SDC, TM-5897/000/00, June 1977.

[4] SCI Publication No. D0.01, "ARC User's Manual Volume I, ARC System Description," Systems Control, Inc., A001-3, 13 August 1976.